TRANSPORT AND SESSION LAYER PROTOCOLS IN COMPUTER NETWORKS

A THESIS

SUBMITTED TO THE FACULTY OF ATLANTA UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE

BY

PETER EJINDU UKUKU

DEPARTMENT OF MATHEMATICAL AND COMPUTER SCIENCES

ATLANTA, GEORGIA

JULY 1985

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

My gratitude to Atlanta University and in particular to the Faculty of Mathematical Sciences cannot be understated. To them I owe everything that is true and honest in this study.

It is very rare to see such combination of individuals who are determined to see to the success of their students without infringement on the quality of the program.

To Dr. Nazir A. Warsi, I thank heartily for his unflinching support and constructive recommendations.

Dr. Benjamin Martin and Dr. Bennett Setzer should cherish my gratitude for their geniality and determination to provide resources essential for success in Computer Science at Atlanta University.

To my lovely wife, Ann, I owe my present success for her inspirational support.

CHAPTER I

PROBLEM DEFINED

The basic premise for the development of Network Architecture can best be described by the terminology of the Organization for International Standards.

The key issues involved in these two layers are naming and addressing of users, connection establishment and termination, buffering and flow control, multiplexing, crash recovery and synchronization. Network Architecture has multiple processes, each of which form a layer. Within the layers are many processes which perform unique functions. These processes must interconnect. The transport layer has multiple desinations, so addressing these destinations constitutes a design consideration. Associated with this problem is how a user knows which server is listening and how the server knows which transport address to listen to. After an established connection has served its purpose, it must be terminated. The mechanism to terminate connections once they are no longer needed constitutes a problem. The method of data transfer is also a matter for consideration. It must be determined whether a simplex or half-duplex transmission method is better than full-duplex transmission.

The mechanism for error-control, error detection, and flow control must be determined and addressed within the transport and session layers. If a message is correctly transferred, the receiver must design a way of acknowledging receipt of the message and telling the sender which message

was received correctly and which was not. The problem of duplicated messages must also be addressed. If a crash occurs, there is the tendency to retransmit the same message again. If a session is timed out, the already transmitted message might reappear--leading to an incorrect connection establishment. If a crash occurs in the middle of a transmission, how could a transmission error be prevented? How can a network avoid loss of data and retransmit the old data?

Some communication channels do not preserve the order in which a message was sent. The transport layer provides a mechanism to allow the receiver to reassemble the pieces of a message. Synchronization is a major design issue in every layer. A fast sender must be stopped from "swamping" a slow receiver. A provision must be made to allow a process to accept long messages. The current status of the receiver must also be know to the sender. Multiplexing presents an issue within the transport and session layers. Channeling many pieces of information through one line demands that many connections must be multiplexed together. A routing decision must also be made when there are multiple paths possible between source and destination. Buffering strategy to use will constitute a design consideration. It is also important to determine whether buffering should be done at both ends. This study will evaluate naming and addressing of users, connection establishment and termination, buffering and flow control, multiplexing, error recovery, and synchronization within the transport and session layers.

CHAPTER II

ASSUMPTIONS FOR THE TRANSPORT AND SESSION LAYER

The advocates of division of network functions into layers maintain that each layer should form an entity which can communicate with the next higher layer. The function performed by each layer should be localized so that the layer could be totally redesigned and its protocols changed in major ways to take advantage of new advances in architectural hardware or software technology without changing the services rendered to the adjacent layers.

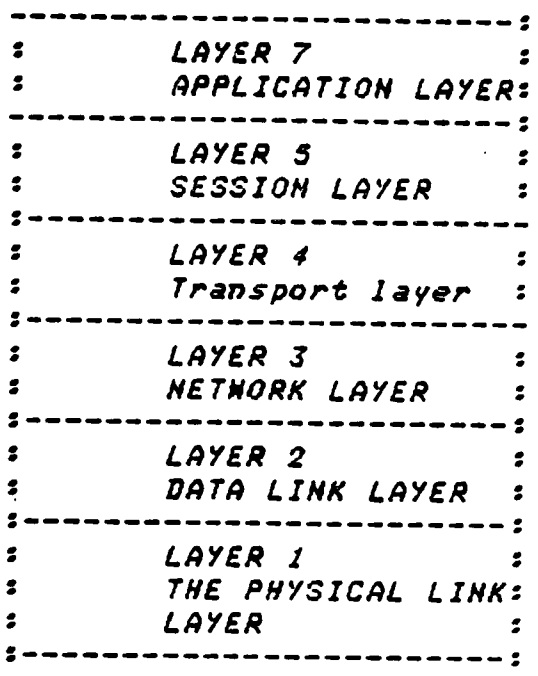
A virtual circuit is assumed mainly due to the adaptability of virtual network to International Standards Organization model. The study assumes an understanding of the International Standards Organization and Organization for International Standards (ISO, OSI) Reference Model.

CHAPTER III

LAYER DEFINITION

The ISO model divides the network functions into layers. The following chart divides the layers into the transmitter and receiver sides and gives the function of each.

	Transmitter	Receiver
Data Link	. Insert frame flags . Provide frame checksum . Insert destination address . Generate next frame number . Provide acknowledgement frames	. Detect frame flags . Verify checksum . Check address . Check frame number . Frame/packet assembly
Network	. Insert packet checksum . Insert destination address . Provide acknowledgement for packets . Generate next packet number . Packet/frame disassembly	. Detect incoming packet . Initiate transfer of packet . Verify checksum . Check address . Decode packet packet control information
Transport	. Insert message checksum . Provide address mapping . Provide acknowledgement of message/package disassembly	. Initiate message . Transfer network . Address mapping message . Decode control message
Session	. Address mapping . Insert control information	. Initiate transfer of message from trans.
Presentation	. Text conversion . Text compression . Encryption	. Text conversion . Text compression . Decoding


LAYER 7
APPLICATION LAYER
LAYER 5
SESSION LAYER
LAYER 4
Transport layer
LAYER 3
NETWORK LAYER
LAYER 2
DATA LINK LAYER
LAYER 1
THE PHYSICAL LINK
LAYER

CHAPTER IV

REVIEW OF THE RELATED LITERATURE

Among the recommendations made by the International Standards Organization was the division of network functions into a series of layers. The ISO OSI standard proposes that each layer should be an entity communicating with other entities. All the entities should exhibit general characteristics such as:

- The interface to the outside world through any number of input/output ports.
- They have a set of private data items.
- The current value of the private data defines the process state.
- When an input occurs the current process state might change and output might be generated.
- Which new state is entered is a function of the input and current state of the process.

Based on the concept of layers the ISO adopted the following principles which should guide the establishment of the layers:

- A layer should be created where a different level of abstraction is needed.
- Each layer should perform a well defined function.
- The function of each should be chosen with an eye towards defining internationally standardized protocols.
- The layer boundaries should be chosen to minimize the information flow across interfaces.

In the following principles laid down by the ISO, the transport and session layer functions are not isolated from the other five layers. Within these two layers are combinations of several interfacing functions with the other layers a network system architecture (see the diagram on the next page).

Mr. C. Folts enumerated the services provided by the transport layer to include the following:

- Connection establishment between one or more session entities within each end-system;
- Data transfer; and
- Flow control: With the flow control service, the session layer can dynamically request to limit data delivered from the transport layer.

The session layer is responsible for the following functions:

- Session-connection establishment and termination, binds and relationship;
- Session-control management enables the presentation entities to determine cooperatively the unique values of the operating parameters;
- User data exchange supports the transfer of a unit of data and prevents the receiving presentation entity from being overloaded with data;
- Data quarantine affords the sending presentation entity explicit control of the data unit to be delivered to the receiving

presentation entity; and

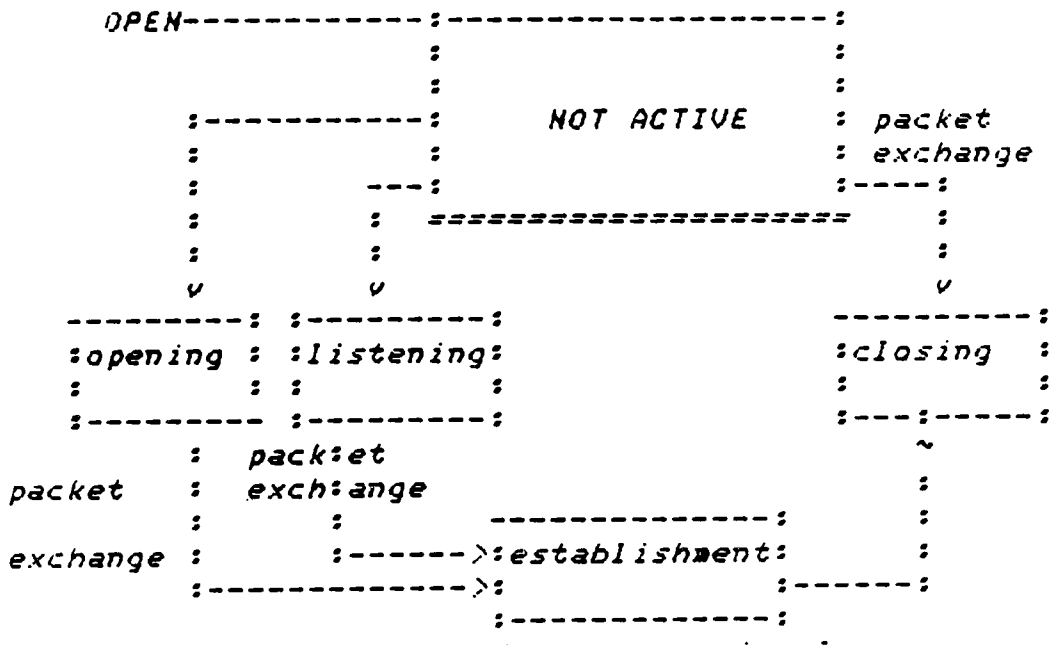
- Interaction management--a dialog control used to establish a two-way simultaneous interaction send only from one end and receive only at the other end.

Connection Establishment

When computer located in different geographically dispersed areas are required to share resources, interprocess communication protocols must be established. The protocols architecture consists of a partially reliable communication service at the lowest level followed by a general purpose fully reliable transport protocol.

The use of different layers and the fact that connection must be established between computers, requires the maintenance of the state information describing the progress of data exchange. The initialization and maintenance of this state information constitute a connection between the two processes provided by the transport program on each side of the connection.

The amount of resources consumed by connection establishment is enormous, so it is advisable to keep the connection open only while processes are actively communicating. The diagram on the next page is a simple connection life-cycle. The source and destination addresses are involved in connection establishment. The source is the local transport address and the destination is the remote transport address. A program has been written to illustrate the action of connection establishment (see the Appendix for program logic and output).


OPEN
NOT ACTIVE
packet exchange
opening
listening
closing
packet exchange
packet exchange
establishment

CHAPTER V

PROGRAM DESCRIPTION

The program at Appendix A demonstrates a simple transport and session layer protocol during connection establishment. In the program, two hosts--A and B--are involved. Host A is requesting for a connection to a resource in host B. The resource and destination addresses are involved in the connection establishment. The source is the local transport address in A. The job of connect is to establish a link between transport addresses.

The action of the transport service is to make connection between two hosts using a set protocol. The program written to illustrate this action is as follow.

Connect

Two parameters are involved in the connect call-local transport address and remote address. The Connect Procedure tries to establish connection between the transport address in the two hosts. A successful connection returns a positive flag which is saved in a local address called Connum. A failure returns a negative flag which is also served in the Connum. In the program each transport address can only make one call. One possible cause of failure is that the address is being used at the present time or the remote may be down or out of service.

Listen

The listen indicates that the remote's request will be accepted. The caller loops until a connection is made. If an unsuccesful connection is made, the caller is suspended.

Send

The send transmits the content of the buffer as a message on the indicated transport connection.

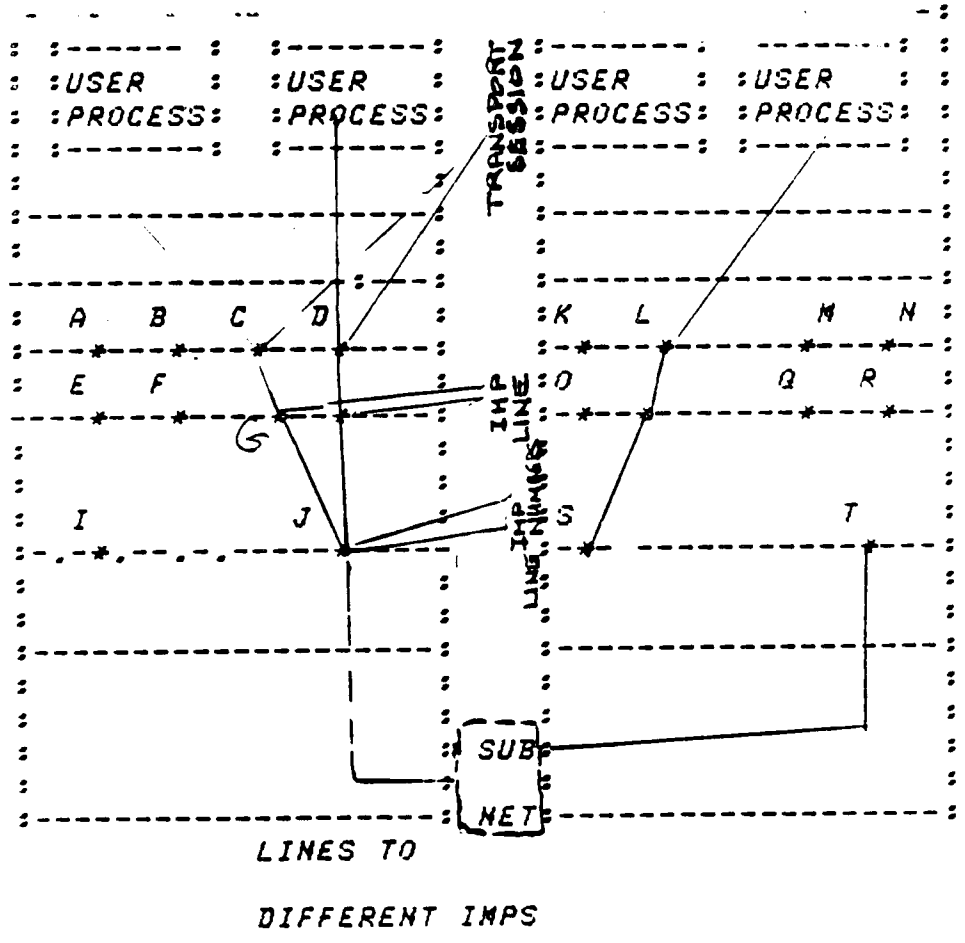
Receive

The receive indicates that the caller is ready to accept the data. It is necessary to state that this program demonstrates an oversimplified transport and session layer protocol. It is not a simulation of a particular system protocol.

CHAPTER VI

SERVICES OFFERED BY THE TRANSPORT LAYER

There are two types of services offered to the user by the Transport layer. A connection-oriented service and a connectionless service. In this type of service a logical connection is maintained for the duration of the transfer (see the diagram below).



The identifiable virtual circuits are GJSP and HJSO. Daniel Schwabe, author of Formal Specification and Verification of a Connection Establishment, described the connectionless mode as a datagram network wherein a unit of data is transmitted as a single unit of event without the overhead of establishing, maintaining, and terminating a logical connection. Schwabe asserted that a connection could be made through service primitives--the duty of the first primitive is to initiate the connection and the other to reject the connections. The initiation of a connection involves naming and addressing of users. First the name of the user will be found and the address determined from the name. A generic name space similar to a central switchboard is used. In a central switchboard, the user dials to the board. The operator intercepts the call and transfers it to the appropriate address. This type of addressing adopts a centralized naming procedure. The major concern during addressing and connection establishment is how a user knows which server is listening to which transport address, and also how a server knows which transport address to listen to. To address this issue, a distinction must be made between name and address. A user uses a process name to identify the address. A connection is established by consulting a name server analogous to the telephone systems directory assistance. There are two types of addressing techniques. The first is hierarchical addressing. This addressing technique uses address space large enough to include the location of the server, while the second, called flat addressing, does not. Flat addressing, as Andrew S. Tanenbaum observed, has no

particular relationship to geography or any other hierarchy. He adduced that:

> Users normally access generic service by name and not by address. The local host must use the name to find transport address of a process willing to do the work. Once address of such a process is known, contact can be established using the connect primitive or its equivalent.

Buffering

The buffer is a temporary storage area for message in a telecommunication network. The buffers are viewed as the transmission storage for handling, queueing, and transferring of message segments. The issue of buffering in transport layer is not what the buffer does or how it can do it efficiently. The main concern is the adequacy of buffer size, who should buffer and how buffers should be created. Mr. Andrew S. Tanenbaum confirmed that the critical issue at this stage is how a fast transmitter can be kept from overrunning a slow receiver. Mr. J. H. Chang identified four main strategies for buffering allocation.

The first is static allocation. In this type of allocation, the longest message is used as a maximum allocation unit. The second type dynamic buffer allocation, allocates buffers to the line when it is requested and deallocates when the message is completely transferred. This method of buffer allocation saves time but can lead to an efficient computer storage. Another type of buffer allocation called semi-dynamic allocation-2, merges static and dynamic allocation methods together. A maximum buffer is allocated but is released immediately after use. The question is raised, as to when buffering should be done.

Andrew S. Tanenbaum replied by stating that if a subnet is unreliable, the sender must buffer all messages sent... if the host wants to offer truly reliable service, using a subnet that can reset virtual circuits at will, the sending host has to buffer all outstanding messages until they are acknowledged by the receiving host.

Dynamic Buffer Allocation

A	Message	B	Comment
1 ——>	< request 8 buffers >	——>	A counts buffers
2 <——	< ack = 15, buf = 4 >	<——	A grants message 0-3 only
3 ——>	< seq = 0, data = m0 >	——>	A has 3 buffers left now
4 ——>	< seq = 1, data = m0 >	——>	A has 2 buffers left
5 ——>	< seq = 2, data - m2 >	....	Message lost but A thinks it has 1 left
6 <——	< lack = 1, buf = 1 buf = 3 >	<——	A acknowledges and 1 permits 2-4
7 ——>	< seq = 2, data = m2 >	——>	A has 1 buffer left
8 ——>	< seq = 4, data = m4 >	——>	A has 0 buffer left and must stop
9 ——>	< seq = 2, data = m2 >	——>	A timeout and transmits
10 <——	< ack = 4, buf = 0 >	<——	Everything acknowledged
11 <——	< ack = 4, buf = 1 >	<——	A may now send 5
12 <——	< ack = 4, buf = 2 >	<——	B found a new buffer somewhere
13 ——>	< seq = 5, data = m5 >	——>	A has 1 buffer somewhere
14 ——>	< seq = 6, buf = 4 >	——>	A is now blocked
15 <——	ack = 6, buf = 4 >	<——	A is still blocked
16	ack = 6, buf = 4 >	<——	Potential deadlock

Legend

Arrows show the direction of transmission

An elipisis (...) indicates a lost message

In the initial stage event A requests for 8 buffers but B gives only three. Event five shows A sending three messages but losing the third one. In event six B acknowledged the receipt of all messages including the sequence number. A is allowed to release buffers and B notifies A to send messages 2, 3, and 4. A discovers that it has already sent messages 2, A waits for the next buffer allocation. Buffer is allocated.

In the event 10, B acknowledged messages but refuses to allow A to continue. Event 16 shows that more buffers have been allocated by B but the message was lost. A become deadlocked because of non-sequencing of the control messages at intervals, giving the acknowledgement and buffer status on each connection. The deadlock is broken eventually.

Multiplexing

Data communications and computer network specialists maintain that the cost of having many virtual circuits is enormous. Some installations bill a user on a connect time and not on the amount of data sent. If connect time is used, the cost for an installation will be heavy. Thus, multiplexing achieved by grouping transport connections according to their destination...(mapping each onto the minimum number of virtual circuits. This type is called upward multiplexing. It reduces performance because the window will usually be full, and users will have to wait their

turn to send one message.

Multiplexing can also occur within the transport layer on band-width basis. Users who frequently use certain band-width can be multiplexed together. Multiplexing by band-width is achieved by having the transport layer open multiple network connections, and distribute the traffic among them on a round-robin basis. This method of multiplexing is called downward multiplexing.

Another group of traffic connections are multiplexed on protocol basis, especially when many networks have to interconnect. Multiplexing can also be done on priority basis. In summary, the whole purpose of multiplexing is either to reduce cost or to improve performance or both.

Flow Control and Deadlock Prevention

There is always a maximum number of messages, adjacent IMP's can exchange. If the sender sends more than the subnet can carry, the subnet will become congested. Andrew S. Tanenbaum, quoting Mr. Belshes, confirmed that the sender dynamically adjusts the window size to match the network's carrying capacity. In order for this technique to work, the carrying capacity is determined in advance. On the other hand, overflow do arise without notice. In local area networks filters are used to discard packets when overflow arose without notice. The filter may also be able to report the loss of packets to their source. This enables the source to recover quickly.

From the brief discussion, it can be asserted that among the main objectives of Flow control are improvement of

performance and deadlock prevention. Some of the prevention objectives are prevention of traffic loss that can occur when a receiver is flooded, avoidance of interference between users and protection of communication network against clogging.

The performance objective include maximize response time in conversational traffic; maximize useful transmission cost; keep the receiver busy at all times. In network environment, several users share computer resources. The contention for this resource can cause congestion and possibly deadlock. Among the set of mechanism for preventing deadlock is end-to-end-flow control. This method of flow control allows the end-point receiver to maintain the sender traffic within limits compatible with the amount of resource available at the receiver end. Another method of controlling traffic is called congestion control. In congestion control, an intermediate sub-system maintains input traffic within limits compatible with amount of resource available to the sub-system.

Mr. L. Pouzin, author of Flow Control in Data Networks Methods and Tools, identified the following methods for controlling congestion within the transport and session layer.

Credit

The sender waits for an indication from the receiver to send a message. In order to prevent congestion, protection is built into the credit system to stop the sender from sending messages forever.

Rate

Pouzin advocated the allocation of slot to control the rate of transmission. The sender transmits at a predetermined rate ... The

rate is supposed to be within the available resources. It can be adjusted dynamically when the message is received.

Delay

The reception of message is delayed but the receiver keeps sending until an acknowledgement is received. When this occurs, the sender timesout after a while and retransmits. The original packet may reappear if it took a different route. Incidentally this results into a duplicate message being sent to the sender. Among the ways suggested for dealing with a delayed duplicate problem is to generate a new address whenever a connection is closed. Another method is never to let the source address be reused, but to permit the reuse of the destination address. A duplicate may be checked by not allowing the combination of address to be linked up again. The sequence number can be made so large that it never recycles.

None of the methods suggested has proven adequate to solve delayed duplicate problem. The following, though advocated, do not prove adequate either.

The first is to restrict subnet design. It means that congestion delays would be sent over the longest possible path. Another approach is putting a hop counter in each packet to enable a data link protocol to discard a packet whose hop counter has exceeded a set maximum. In the third approach an IMP is to be synchronized.

Among all three methods advocated, the one which has called most attention is the clocking method. A host can set a time Y which controls the use of a connection identifier. If a connection is closed, the clock is allowed to tick until a time Y is reached, then the same connection identifier can be reused. The host can only keep information about old duplicate and closed connections for a set period after closing. The added burden here is that the host must also remember the last sequence number used for each connection for a time T, beyond when the connection was closed. Tomlinson proposed a synchronization method to get around a host losing all memory where it was after crash. A host can be equipped with the time of the day clock. An initial time or specific time in which a sequence number must be set. The forbidden region in the figure shows an illegal sequence number. Any sequence number in the forbidden region is to be discarded. The Transport station reads the forbidden region to check if an initial sequence number matches the one in the forbidden region. If it matches, the message is discarded (see the figure below). Resynchronization problem does not solve the problem of delayed duplicate completely. If a message which was thought to be lost suddenly shows up, an incorrect connection would be made. Tomblinson advocated a three-way handshake to solve the problem. Global clocking method need not be used here, since the hosts are not supposed to start sending any message with the same sequence number. Synchronization method can be applied here. Tomlinson illustrated by assigning a sequence number to A which A sends to B. B then acknowledges with SYSN2, indicating that the message J sent by A has been received. B then

announces its own initial sequence number L. A then acknowledges B's message that means B's choice of initial sequence number in the first data message.

(A)

#	A	Packet	B	Comment
1	Type = SYN1, seq = j	:	A	Wishes to initiate connection
2	Type - SYN1, seq = y	:	B	Accepts A's request
3	Type = data, seq = x, ack = Y	:	A	Acks and starts

(B)

#	Packet	B	Comment
1..	Type = SYN1, seq = y, ack = x	:	Delayed duplicate arrives at B
2	Type = SYN2, seq = y	:	B accepts A's request
3	Type = reject, ack = y	:	A rejects B's connection

(C)

#	A	Packet	B	Comment
1		: Type = SYSN1, seq = x	:	Delayed duplicate SYN arrives at B
2		: Type = SYN2, seq = y	:	B accepts A's request
3		: Type = SYN1, seq = y, ack = z	:	Delayed duplicate B sees Z and Y
4		: Type = reject, ack = y	:	A rejects B's as before

Tomlinson referred to the Figure (A) as the normal operation, (B) delayed SYN1 and (C) delayed SYN1 and acknowledgement. The delayed duplicate is shown by (...). This duplicate got to B with A's knowledge. B responds by sending a SYN2 message. A was trying to set up a new connection. When B discovers that the message was a delayed duplicate, it abandons it. In Figure (C) both delayed duplicate and acknowledgement are floating around in subnet. B originally used Y as the initial sequence number but incidently another sequence number Z showed up at B. A acknowledges Z but B realizes that a duplicate is floating around, since it was expecting Y to be acknowledged. B terminates the connection. To solve this problem, Tomlinson adovated that each host should send a

CLOSE request and then wait until it has received both an acknowledgement and in sequence CLOSE request from the other side.

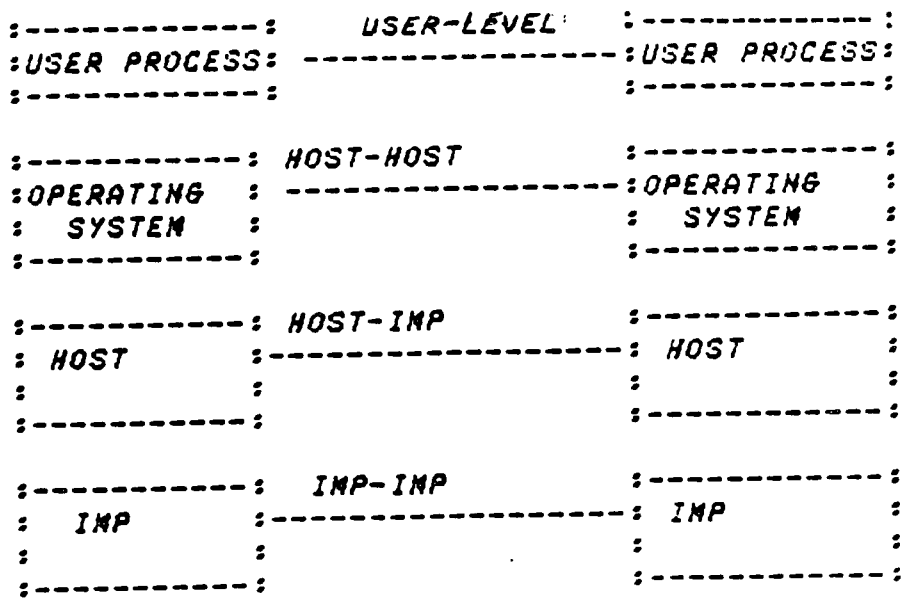
Crash Recovery

If a host crashes in the midst of a transmission, it may lose information about where it was before the crash when it restarts. When the host comes back up, its tables are reinitialized so that it no longer knows precisely where it was. The receiver might send a broadcast message to all the nodes asking for the status of all open connections. The sender can retransmit the message. The receiver can be programmed to retransmit the last message, or to retransmit only if a message is outstanding. A crash can also occur in the subnet. The Transport protocol may be designed to allow the host to inquire about the status of the other. If the host inquires and discovers that the subnet has crashed, the host can find out the appropriate message and retransmit it. This can only be possible if the host has kept copies of the messages. But if the host assumes that the subnet kept a copy, it cannot recover by transmission of the message. Tomlinson concluded that no matter how the sender and the receiver are programmed, there are always situations where the protocol fails to recover properly.

CHAPTER VII

IMPLEMENTATION OF THE TRANSPORT LAYER IN ARPARENT

In Arparnet, the network control protocols handle connection establishment, management, and deletion of end-to-end connection. Network control protocol was designed by DARPA to act as a perfect subnet without any relationship with an unreliable datagram. DARPA's desire to interconnect NCP with an unreliable datagram lead to the development of Transmission control protocol. In the figure below, the host-to-host communication is handled by the Network Control Program (NCP), which is the operating systems. The NCP can provide both simplex and full-duplex connection.



The host-to-host connection are done on an odd-even basis. There are thirty-two-bit numbers in a port. Even numbers are used for receiving, any odd numbers for sending.

During connection the STR (sender to receiver) is exchanged for RTS (receiver to sender). There is one even and one odd port in any connection. The gender is the low-order bit of the port. When a connection is being initiated, the INI primitive is used. This primitive allows a user to wait for a connection until a LISTEN primitive specifies a local port. When a user requests a connection by CALL INIT, the Transport station checks the gender of the port to find out if there is any outstanding message either outbound or inbound. Based on the status of the message, the Transport station either queues the message for a while or uses CLS (CLOSE) to close the message. The connection establishment in ARPARNET is not done until a buffer is allocated. An explicit buffer allocation which uses three control message: ALL (ALLOCATE), GVB (GIVE BACK), and RET (RETURN) is used. No message is sent until the host indicates a readiness to receive by sending all messages. Static buffer allocation strategy is used. The message limit and the bit limit is specified. The GVB parameter is used to reclaim an allocated buffer which was not used. There are two interrupt mechanisms, INR and INS, provided by the NCP. If a host wants to check the status of another host, it uses the ECHO message. An ECHO reply (ERP) is used to respond to a request for transmission. An error can be detected by error message. The message enables the network operator to debug the software.

TCP Transmission Control Protocol

Mr. C. A. Sunshine and Y. K. Dalal proposed that the TCP was designed with strong worst case assumptions about underlying transmission medium. Sunshine and Dalal were referring to the original assertion by ARPANET that the transmission control protocol was designed to support an unreliable Datagram. ARPANET assumed that packets could be damaged, lost, duplicated or delivered out of sequence, thus they suggested the connection of the Transmission control protocol to a Datagram service. In this set up initial sequence numbers are used to validate arriving packets. Delayed duplicate problem are handled by a connection reset process simply called RTS. Variables-size sliding window protocol are used to handle flow-control problems.

CHAPTER VIII

IMPLEMENTATION OF TRANSPORT AND SESSION LAYER IN SNA

The path control and transmission control layers are responsible for the implementation of transport and session layers in SNA. It is the responsibility of path control to route messages to the correct active half-session or to communicating session control in the destination node. R. J. Cypser, author of Communications Architecture for Distributed Systems, described the path control as a cargo in a trucking firm. The cargo containing messages are packaged and routed to the next node.

Addressing in SNA are assinged to each systems service control points (SSCP) which is a special purpose network addressable unit used for network management. The SSCP does the overall management of the network, such as bringing up the network, helping to establish logical connections between network addressable units, recovery and maintenance.

A connection is established by a process requesting the SSCP managing its domain to set up session. The addressing of the connection are done by the areas and elements. Within the addressing area are a number of subareas. To get a network addressable unit, a combination of subareas and element addresses are used. When a session is started, the session control oversees the activities. The session does the activation and deactivation of one-half-session, opening and closing of flow gate. It can start a traffic and clear the data traffic as directed by the end-user at the primary network addressable unit.

The session control can help a higher level resynchronizatize if a sequence gets lost or gets into trouble.

CHAPTER IX

CONCLUSIONS

Principles of layered architecture is still an area where a lot of research remains to be done. Most of the techniques used to deal with crash recover, network congestion, deadlock, synchronization, multiplexing and buffering do not provide a perfect solution to the problem. ARPANET, SNA and DECNET have implemented techniques for addressing the problems mentioned above but much still need to be done to provide a perfect recovery technique.

A better crash recovery strategy would be a technique which improves upon the global clocking method, use of sequence number and transmission time.

BIBLIOGRAPHY

Canesch, F. "Role of the Session Layer in OSI Architecture." Communications Engineering International, Vol. 5, No. 2.

Cypser, R. J. Architecture for Distributed Systems. Philippines: Addison Wesley Publishing Company, Inc., 1978.

David, Flint, C. An Introduction to Local Area Network. New York: John Wiley and Sons.

Emons, W. W. "OSI Session Layer: Services and Protocol." Proceedings Institute of Electrical Engineering (IEEE), Vol. 71, No. 12 (December 1983).

Emmons, W. F. and Chandler, A. S. "OSI Session Layer: Services and Protocols." Proceedings Institute of Electrical Engineering, Vol. 1, No. 12 (December 1983).

Folts, H. C. "Procedure for Circuit-Switched Service in Synchronous Public Data Network." Institute of Electrical Engineering Transaction on Communication, Vol. Com-28, No. 4 (April 1980).

Korfhage, Robert. Computer Networks and Communication. New Jersey: Afips Press, 1978.

Lam, S. Simon and Reiser, Martin. "Congestion Control of Store and Forward Networks by Input Buffer Limits: An Analysis." Institute of Electrical Engineering, Vol. 71, No. 12 (December 1983).

Mimica, Oscar and Marsden, Brian. "A Datagram-Based Network Architecture for Microcomputers." Institute of Electrical Engineering, Vol. 71, No. 12 (December 1983).

Naffah, N. "Diagnostics and Supervision in Informatics Networks." Communications Engineering International, Vol. 5, No. 2.

Pouzin, L. "Flow Control and Methods and Tools." Proceedings of the International Conference on Computer Communication (eds.). Toronto, August 1976.

Schwartz, Mischa and Thomas, Sterne E. "Routing Techniques Used in Computer Networks." Institute of Electrical Engineering Transaction on Communications, Vol. Com-28, No. 4 (December 1984).

Schwade Daniel. "Formal Specifications and Verification of a Connection Establishment Protocol." Institute of Electrical Engineering Transaction Communications, Vol. Com-27, No. 4 (December 1984).

Sunshine, A. Carl and Yogen, Dalal K. "Connection Management in Transport Protocols." Computer Networks (January 1984).

Tanenbaum, Andrew S., Computer Networks. California: Prentice-Hall, 1981.

APPENDIX

Program Listing

```
program simulate transport (input, output);

{*************************************************************************
* This program demostrates a simple transport layer protocol during      *
* connection establishment.                                              *
* In this program two host A and B are assumed. Host A is request-       *
* ing for a connection to the resource in host B. The source and         *
* destination addresses are involved in the connection establishment.    *
* The source is the local transport address in A. The job of connect     *
* is to establish a link between the two transport addresses.            *
* The listen indicates that theuser technically referred to the          *
* Caller is willing to accept connection directed at the indicated       *
* address. When a transport connection is no longer needed the CLOSE     *
* CALL terminates it. Another subsection, the send call transmits        *
* the contents of the buffer as a message on the indicated transport     *
* connection while the RECEIVE CALL indicates the caller's desire to     *
* accept.                                                                *
*************************************************************************}

  const
        n                       = 8;

  maxconn                       = 28;
  maxmsg                        = 28;
  maxpkt                        = 28;
  queue0                        = 0;
  queue1                        = 1;
  message0                      = 0;
  message1                      = 1;
  credit                        = 28;
      ok;                       = 0;
  errful                        = -1;
  errreject                     = -2;
  errclosed                     = -3;
  lowerr                        = -4;
  type
  node1                         = array [1..n] of array [1..maxconn] of
                                  integer;
  node2                         = array [1..n] of integer;
  node3                         = array [1..maxconn] of boolean;
  node4                         = array [1..5,1..n] of integer;
```

```
bit                              = 1..1;
transportaddress                 : 1..maxconn;
     connid                      = 1..maxconn;
pkttype                                                    =
                                 (callreq,callacc,clearreq,clearconf,
                                  datapkt, credit);
cstate                                                     =
        (idle,listening,waiting,queued,open,sending,receiving,closing);
message                          = array [0..maxmsg] of 0..225;
msgptr                           = ^message;
errorcode                        = lowerr..maxconn;
pktlength                        = 0..maxpkt;
packet                           = array [pktlength] of 0..225;
     var
      countloop                  : integer;
      wakeup1                    = array [1..n] of integer;
      sleep1                     : array [1..n] of integer;
      receive1                   : array [1..n] of integer;
      send1                      : array [1..n] of integer;
      listen1                    : array [1..n] of integer;
      connect1                   : array [1..n] of integer;
      packetar                   : array [1..n] of integer;
             a                   : node1;
             b                   : node2;
             c                   : node2;
           col                   : integer;
        testcol                  : integer;
        count                    : integer;
            i                    : integer;
            p                    : integer;
        match                    : boolean;
          det                    : node3;
         perms                   : node4;
           row                   : integer;
           row2                  : integer;
 listenaddress                   : transportaddress;
     listconn                    : connid;
         data                    : packet;
         conn                    : array [connid] of record;
  localaddress                   : transportaddress;
     msgsent                     : integer;
  msgreceived                    : integer;
    state                        : cstate;
  usebufferaddress               : msgptr;
     bytecount                   : 0..maxmsg;
  clrreqreceived                 : boolean;
         timer                   : integer;
```

```
        credits                 : integer;
          end;

*************************************************************************
*                                                                       *
*  This procedure generates random number in the interval 0, 1          *
*  The random number is used to simulate the average arrival            *
*  reception rate per request.  A period of 1000 is chosen and          *
*  the output is generated every 200 minutes                            *
*                                                                       *
*************************************************************************

function random (var seed : integer) : real;

 begin
   seed                         : = (25173 * seed + 13849) mod 65536;
   random                       : = seed / 65536;
   end;

*************************************************************************
*                                                                       *
*  This generates the assumed state of the host by random               *
*  selection.                                                           *
*                                                                       *
*************************************************************************

procedure fromnet;
  var
     n                          : integer;
   col                          : integer;
  row                           : integer;
maxconn                         : integer;
  a                             : array [1..8,1..28] of integer;
  row1                          : integer;
  row2                          : integer;
  begin
      n                         : = 8;
maxconn                         : = 28;
for row : = 1 to n              do
for col : = 1 to maxconn do
  a [row,col] : = 0;
         col  : = 0;
for row1      : = 1 to 7 do
for row2      : = row1 + 1 to n do
begin
    col       : = col + 1;
    a [row, col] : = 1;
    a [row2, col] : = 1;
    end;
writeln ('************       INITIAL   STATE   OF   HOST  B
*******************');
```

```
writeln(' ');
for col       : = 1 to maxconn     do
write(col,:2);
writeln('  ');
for row : = 1 to maxconn   do
write (a [row,col] : 2);
writeln ('  ');
end;
procedure evaluatenodesinb;
var
  t                              : integer;
  row1                           : integer;
  maxconn                        : integer;
  a                              : integer;
  row2                           : integer;
  col                            : integer;
    c                            : integer;
  match                          : boolean;
    row                          : integer;
    det                          : array [1..28] of boolean;
      P                          : integer;
  count                          : integer;
  testcol                        : integer;
        n                        : integer;
     perms                       : array [1..5,1..28] of integer;
         b                       : array [1..8] of integer;
    begin
        n                        : = 8;
    maxconn                      : = 28;
    for col                      : = 1 to maxconn do
    a [row,col]                  : = 0;
    col                          : = 0;
    for row1                     : = 1 to 7 do
    for row2                     : = row1 + 1 to 8 do
    begin
       col                       : = col + 1;
    a [row,col]                  : = 1;
    end;
    for row                      : = 1 to 5 do
    for col                      : = 1 to n do
    read (perms [row,col]);
    for i                        : = 1 to maxconn do
    det [i]                      : = false;
    count                        : = 0;
    col                          : = 0;
```

```
while (col < maxconn) and (count < maxconn)
do
 begin
     col            : = col + 1
       if not det [col]
       then
          begin
             det [con]      : = true;
             count          : = count + 1;
                 p          : = 0;
                 while (p < 5) and (count < maxconn)
                 do
                  begin
                      p =   : = p + 1
                  for i : = 1 to n do
                      b [i]       : = a [perms [p,i],col];
                  for testcol     : = col + 1 to maxconn do
                      if not det [testcol]
                      then
                         begin
                         for i : = 1 to n do
                         c [i] : = a [i,testcol];
                           if match
                           then
                              begin
                              det [testcol] : true;
                           count     : = count + 1
       write ('node#':8,testcol:3,'cannot be used':18);
       end
          end
            end
              end
         end;
```

```
*************************************************************************
*                                                                       *
*                                                                       *
*                          LISTEN PROCEDURE                             *
*                                                                       *
*************************************************************************


procedure listen;
var
  t                              : integer;
 listenaddress                   : integer;
 listenn                         : integer;
 found                           : boolean;
 i                               : integer;
 listenconn                      : integer;
 data                            : array [0..255] of integer;
 conn                            : array [0..30] of integer;
 open1                           : integer;
 queued                          : integer;
 n                               : integer;
 maxconn                         : integer;
 begin
    found
    while (i < = maxconn) and not found
    do
      begin
          if (conn [i] = queued) and (conn [i] = t)
          then
             found                : = true
          else
                 i                : = i + 1;
                 if not found
                 then
                    begin
                        listenaddress     : = t;
                        i                 : = listenconn
                        end;
                        conn [i]          : = open1;
                        conn [i]          : = 0;
                        listenn           : = 0;
                 end;
                    end;
```

```
*****************************************************************************
*                                                                           *
*                                                                           *
*                                  CONNECT                                  *
*                                                                           *
*                                                                           *
*****************************************************************************


procedure connect;
   var
     open                          : integer;
     waiting                       : integer;
           t                       : integer;
           lr                      : integer;
           i                       : integer;
           r                       : integer;
        conn                       : array [1..28] of integer;
   maxconn                         : integer;
   msgsent                         : integer;
   msgreceived                     : integer;
   localaddress                    : integer;
   remoteaddress                   : integer;
   clrreqreceived                  : boolean;
       credits                     : integer;
       timer                       : integer;
       connect1                    : integer;
       errejject                   : integer;
       state                       : integer;
       idle                        : integer;
       errfull                     : integer;
         data                      : array [0..255] of integer;
         begin
             i                     : = maxconn;
             data [0]              : = r;
           while (conn [i] = idle
           then
              begin
                  msgsent          : = 0;
               msgreceived         : = 0;
          localaddress             : = 1;
          remoteaddress            : = r;
            state                  : = waiting;
          clrreqreceived           : = false;
                credits            : = false;
                timer              : = 0;
                end;               : = 0;
                  if clrreqreceived
                  then
                     begin
                          connect1    : = errject;
                           state      : = idle;
```

```
                                end
                                  else
                                     begin
                                      writeln;
                                      connect1  : = errfull;
                end;
*****************************************************************************
*                                                                           *
*                                                                           *
*                                SEND                                       *
*                                                                           *
*****************************************************************************

procedure send;
             var
              cid                 : integer;
              open1               : integer;
              data                : array [0..255] of integer;
             bufptr               : array [1..28] of integer;
             bytes                : integer;
                 i                : integer;
             count                : integer;
                 m                : integer;
          bytescount              : integer;
            maxpkt                : integer;
          msgsent                 : integer;
           send                   : integer;
           state                  : integer;
           sending                : integer;
           bytecount              : 0..255;
           clrreqreceived         : boolean;
           credits                : integer;
                 ok               : integer;
           errclosed              : integer;
           begin
               state              : = sending;
           bytecount              : = 0;
           if (not clrreqreceved) and (credits = 0)
           then
              errclosed           : = 0;
              if not clrreqreceived
              then
                 begin
                     repeat
                          if bytes - bytescount > maxpkt
                          then
                          begin
```

```
                              count    : = maxpkt;
                              m        : = 1;
                              end
                                 else
                 count        : = bytes  -  bytescount;
                     m        : = 0;
                 for i : = 0 to count  -  1 do
                data [i] : = bufptr [bytecount + 1];
                bytecount     : = bytecount + count;
                end;
                  until
                      bytecount      : = bytes;
                      msgsent        : = msgsent + 1;
                      credits        : = credits - 1;
                      send           : = ok;
                      end
                        else
                           begin
                               send : = errclosed;
                               state : = open1;
                               end;
                                 end;
```

```
*************************************************************************
*                                                                       *
*                                                                       *
*                                RECEIVE                                *
*                                                                       *
*************************************************************************
```

```
procedure receive;
         var
            i                   : integer;
         bytes                  : integer;
         conn                   : array [1..30] of integer;
   clrreqreceived               : boolean;
         state                  : integer;
      receiving                 : integer;
 userbufferaddress              : integer;
       bufptr                   : integer;
      bytecount                 : integer;
      msgreceived               : integer;
      receive                   : integer;
      errclosed                 : integer;
listenaddress                   : integer;
      data                      : array [0..255] of integer;
      begin
           if not clrreqreceived
```

```
          then
             begin
                 state          : = receiving;
         userbufferaddress      : = bufptr;
             bytecount          : = 0;
             data [0]           : = 0;
             data [i]           : = 1;
       msgreceived              : = msgreceived + 1;
         bytes                  : = bytecount
             end;
             if clrreqreceived
             then
                receive         : = errclosed
              else
                 receive        : = ok;
                 end;
```

```
*************************************************************************
*                                                                       *
*                                                                       *
*                          PACKETARRIVAL                                *
*                                                                       *
*                                                                       *
*************************************************************************
```

```
var
 count                          : integer;
 listenaddress                  : integer;
 localaddress                   : boolean;
 listenconn                     : integer;
 timeout                        : integer;
 msgreceived                    : integer;
 bytecount                      : integer;
 data                           : array [0..255] of integer;
 queued                         : integer;
 userbufferaddress              : array [1..8] of integer;
 timer                          : integer;
 msgsent                        : integer
 idle                           : integer;
 closing                        : integer;
 waiting                        : integer;
 receiving                      : integer;
 sending                        : integer;
 state                          : integer;
 open1                          : integer;
 callreq                        : integer;
 remoteaddress                  : integer;
 clrreqreceived                 : boolean;
 i                              : integer;
```

```
cid                                 : integer;
q,m                                 : integer;
ptype                               : integer;
begin
    remoteaddress                   : = data [1];
    if localaddress
    then
       begin
           listenconn               : = cid;
           state                    : = open1;
           end
    else
       begin
           state                    : = queued;
           timer                    : = timeout
           end;
           msgsent                  : = 0;
           msgreceived              : = 0;
               state                : = open1;
           clrreqreceived           : = true;
           if state                   = closing
           then
              begin
                  state             : = idle;
                  timer             : = 0;
           end;
for i : = 0 to count - 1 do
begin
userbufferaddress [bytecount + 1] : = data [i];
bytecount                           : = bytecount + count;
end;
  if m = 0
  then
     timer = 0;
     end;
```

```
*************************************************************************
*                                                                       *
*                                                                       *
*                                CLOCK                                  *
*                                                                       *
*                                                                       *
*************************************************************************
```

```
procedure clock;
var
  conn                              : array [1..28] of integer;
```

```
    n                                : integer;
 seed                                : integer;
 serve                               : integer;
 queue                               : integer;
 wait                                : integer;
 count                               : integer;
 time                                : integer;
 timer                               : integer;
 level                               : real;
 ran                                 : real;
 ave                                 : real;
   r                                 : integer;
   i                                 : integer;
 maxconn                             : integer;
begin
    n                                : = 5;
    level                            : = n * 0.01;
    seed                             : = 1;
    serve                            : = 0;
    queue                            : = 0;
    wait                             : = 0;
   count                             : = 0;
   for time : = 1 to 1000 do
   begin
       serve                         : = serve - 1;
       ran                           : = random (r);
       if ran                        < = level
       then
           count            : = count + 1;
           wait             : = wait + serve + 10 queue;
           end;
             if (serve = 0) and (queue > 0)
             then
                serve     : = 10;
                queue     : = queue - 1;
                end;
 if ((time) mod 200 = 0) and (count <> 0)
 then
    ave
    end;           : = wait / count;
      end;
```

```
*************************************************************************
*                                                                       *
*                                                                       *
*                                SLEEP                                  *
*                                                                       *
*************************************************************************

procedure sleep;
  var
    counterloop                 : integer;
    time                        : integer;
    begin
        time                    : = time - 1;
    loopcount                   : = loopcount + 1;
    end;
end;

*************************************************************************
*                                                                       *
*                                                                       *
*                               WAKEUP                                  *
*                                                                       *
*************************************************************************

procedure wakeup;
  var
    open                        : integer;
    queued                      : maxconn;
    begin
         if m                   = 0
         then
            open                : = open - 1;
            end;
begin (*  main line                                                    *)
connect;
listen;
sleep;
wakeup;
send;
receive;
packetarrival;
evaluatenodeb;
packet;
clock;
writeln ('n', 'time', 'count', 'wait', 'ave');
loopcount    : = 15;
while loopcount < = 1 do
 begin
     loopcount      : = loopcount - 1;
     tonet;
     end
end.
```